

Balanced Dynamic Content Addressing in Trees*

Stefanie Roos[‡], Martin Byrenheid[†], Clemens Deusser[†], Thorsten Strufe[†]

[‡]University of Waterloo

sroos@uwaterloo.ca

[†]TU Dresden

{firstname.lastname}@tu-dresden.de

Abstract—Balancing the load in content addressing schemes for route-restricted networks represents a challenge with a wide range of applications. Solutions based on greedy embeddings maintain minimal state information and enable efficient routing, but any such solutions currently result in either imbalanced content addressing, overloading individual nodes, or are unable to efficiently account for network dynamics.

In this work, we propose a greedy embedding in combination with a content addressing scheme that provides balanced content addressing while at the same time enabling efficient stabilization in the presence of network dynamics. We point out the trade-off between stabilization complexity and maximal permitted imbalance when deriving upper bounds on both metrics for two variants of the proposed algorithms. Furthermore, we substantiate these bounds through a simulation study based on both real-world and synthetic data.

I. INTRODUCTION

Efficiently routing packets while maintaining little to no state information is a fundamental problem of networking. The issue concerns Internet routing, in particular content-centric networking [1], as well as dynamic networks such as wireless sensor networks [2] and Friend-to-Friend (F2F) overlays in the manner of Freenet [3]. The routing configuration is frequently adapted to implement content addressing, where the node identifier (or: address) is used to determine the allocation of resources to specific nodes. This scenario typically makes the configuration and routing particularly difficult, as the nodes are expected to exhibit extensive dynamics in terms of joining and leaving the system.

Greedy embeddings guarantee the success of stateless greedy routing and thus facilitate efficient communication [4]. All existing distributed greedy embeddings are based on creating a spanning tree and subsequently assigning identifiers to each node. Some embedding algorithms can account for topology changes without a complete recomputation of the local state [5], [6]. In contrast to structured P2P overlays, greedy embeddings do not require the ability to change the network topology, making them suitable for all of the above scenarios.

Implementing content addressing on greedy embeddings, however, faces several challenges. The current proposals are either unable to assign content in a fair manner [7], [8], are unable to deal with dynamics [1], or considerably reduce the efficiency by establishing an additional overlay [9].

We aim to realize fair resource allocation in terms of a balanced content addressing in such trees in dynamic environments. In other words, we require an embedding algorithm in combination with a content addressing scheme such that i) the overhead of stabilization after node arrivals or departures is low on average and ii) the content addressing is balanced, i.e., the fraction of content assigned to a node should not considerably exceed its share of the overall storage capacity.

In this paper, we propose to assign each content an address in the form of a vector of keyed hashes. Similarly, we assign node addresses in the form of vectors. The vector encodes the part of the namespace (in our case: hashes of content) that is allocated to the respective node, and each component of the vector contains a tuple indicating ranges within the namespace. Node addresses are only changed if topology adaptations result in nodes being responsible for more addresses than the current upper bound permits.

Our algorithm assigns at most $\mathcal{O}\left(\frac{\log n}{n}\right)$ of the content to a node at any time if the tree depth is $\mathcal{O}(\log n)$. Thus, the asymptotic bound matches the bound for DHTs [10]. Furthermore, the expected communication complexity for stabilization after a node join or departure is $\mathcal{O}(\text{polylog}(n))$ if the expected number of siblings, i.e., the nodes with the same parent, is bound polylog in n . Otherwise, if such a bound on the number of siblings does not exist, the use of virtual binary trees allows us to achieve polylog complexity nevertheless, at the price of storing up to $\mathcal{O}\left(\frac{\log^2 n}{n}\right)$ of the content on one node. We perform a simulation study based on real-world churn traces and topologies of several thousands of nodes to quantify the stabilization overhead and the balance of the content addressing in exemplary scenarios. Our results indicate that i) the average stabilization overhead is reduced to less than 3% of the overhead of a complete re-embedding, and ii) the content addressing exhibits a similar or even better fairness than common content addressing schemes such as DHTs.

II. RELATED WORK

Greedy embeddings assign coordinates to nodes in a graph such that nodes can route messages based only on the coordinates of their neighbors. Generally, an embedding algorithm computes such an embedding by first constructing a spanning tree and then assigning coordinates starting from the root. Parents assign their children coordinates based on their own coordinate. In this manner, greedy embeddings realize efficient

*Extended Version of 'BD-CAT: Balanced Dynamic Content Addressing in Trees', INFOCOM 2017

routing in any connected graph while maintaining very little state information.

During the last years, a multitude of embedding algorithms has been developed, using coordinates from hyperbolic [5], [7], [11], [12], Euclidean [12], [13], or custom-metric spaces [6], [14]. However, the problem of content addressing is mostly disregarded, with a few notable exceptions discussed in the following.

For instance, the authors of [7] and [15] show that their embedding allows for content addressing. However, neither consider the fraction of addresses, and thus content, assigned to individual nodes. When applying [7] on autonomous system (AS) topologies of several hundreds of nodes, the algorithm allocates more than 90% of all content to one node [1].

To the best of our knowledge, [8] first considers load balancing in terms of content addressing for greedy embeddings. The authors design Prefix Embedding, an embedding algorithm for Friend-to-Friend (F2F) overlays, and evaluate how their design performs when applied for content storage and retrieval in such route-restricted overlays. Their simulation indicate a high imbalance with regard to the fraction of stored content, sometimes assigning more than 50% of all content to a single node in an overlay of tens of thousands of nodes.

Roos et al. [1] inversely adapt the addressing scheme for the content and assign *topology-aware keys* to files, i.e., the address of a file depends on the structure of the spanning tree. In this manner, the expected fraction of files with an address in a certain range corresponds to the fraction of node coordinates in this range. Though the content addressing is indeed balanced, the approach requires that the spanning tree is globally known. Furthermore, network dynamics result in constant changes of node coordinates and file addresses, which make indexing of addresses difficult.

In contrast, [9] circumvents the problem of content addressing directly on the embedding by establishing an additional structured overlay on top of it. However, in this manner, they decrease the efficiency of the routing by a factor of about 4.

In summary, balanced content addressing in embeddings for dynamic networks without global topology information is an open problem. In the following, we propose and evaluate a solution.

III. PROBLEM FORMALIZATION

In this section, we introduce basic notation and formally express our goals. The key terms we need to define are those of a (greedy) embedding, a content addressable storage, and a stabilization algorithm for such a structure.

A. Graphs and Embeddings

Throughout the paper, we rely on a graph $G = (V, E)$ with nodes V and edges $E \subset V \times V$. For simplicity, we restrict our analysis to graphs that are bidirectional, i.e., $(u, v) \in E$ iff $(v, u) \in E$ for all $u, v \in V$, and connected, i.e., there exist $w_0 = u, w_1, \dots, w_{l-1}, w_l = v$ such that $(w_{i-1}, w_i) \in E$ for all $i = 1 \dots l$. Furthermore, we denote the set of *neighbors* of $v \in V$ by $N(v) = \{u \in V : (u, v) \in E\}$.

A *spanning tree* is defined as a subgraph $T_G = (V, E^T)$ of G such that T_G is connected and $E^T \subset E$ is of minimal size. In a spanning tree, there exists exactly one path between every source node s and destination e . A *rooted spanning tree* is a spanning tree T_G with a distinguished element $r \in V$, the root. We express the positions of nodes in the spanning tree with regard to the root. The *level* or *depth* of a node u is the length of the unique path between u and the root in the spanning tree. Furthermore, the depth of the tree is the maximal depth over all nodes. In addition, the relation of two nodes $u, v \in V$ can be expressed in relation to the root. If u is included in the unique path between the root r and v , u is an *ancestor* of v and v a *descendant* of u . Furthermore, if the edge $(v, u) \in E^T$, u is the parent of v and v a child of u . Children of the same node are called *siblings*. Embeddings usually rely on rooted spanning trees to assign coordinates to nodes.

Definition III.1. A (graph) embedding on a graph $G = (V, E)$ is a function $id : V \rightarrow \mathbf{X}$ into a metric space (\mathbf{X}, d_X) . We call $id(u)$ the coordinate or address of u . Consider a pair of distinct nodes $u, v \in V$, $(u, v) \notin E$. The embedding id is called *greedy* if for all such pairs, u has a neighbor $w \in N(u)$ with $d_X(id(w), id(v)) < d_X(id(u), id(v))$. The algorithm **A** for deriving the embedding id is called an embedding algorithm.

For brevity, we generally write distance of u and v to refer to the distance of their coordinates. The above definition holds for any distance $d_X : \mathbf{X} \times \mathbf{X} \rightarrow \mathbb{R}$. We introduce realizations for \mathbf{X} and d_X in Section IV. Then, an equivalent definition of a greedy embedding is the guaranteed successful termination of the standard greedy routing algorithm, which specifies that each node along the path between source and destination forwards the message to the closest neighbor to the destination. If the coordinate assignment id relies on the previous construction of a rooted spanning tree, we call id a *tree-based embedding* or *tree-based greedy embedding* if id is greedy. So, greedy embeddings allow the discovery of a node by a standard greedy algorithm. However, there is little work on how to store and retrieve content based on such an embedding.

B. Balanced Content Addressing

Content addressing generally refers to a deterministic addressing scheme for content. In the context of distributed systems, content addressing implies mapping content to nodes based on node coordinates and content addresses. Here, we map content to the node closest to the content's address. The scenario can be easily generalized such that content is stored on $k > 1$ nodes by e.g., storing content on the closest k nodes or using k different addresses for each file [16].

In order to allow for content to be stored on closest nodes, we first need to extend the notion of a greedy embedding.

Definition III.2. Let $id : V \rightarrow \mathbf{X}$ be a greedy embedding on a graph G and $\mathbf{X}' \subset \mathbf{X}$ a countable address space, and

$ca : C \rightarrow \mathbf{X}'$ an addressing function for a set of content C . Then id is called a content addressable greedy embedding if i) $|M(x')| = |\argmin_{v \in V} \{d_X(id(v), x')\}| = 1$ for all $x' \in \mathbf{X}'$, and ii) $\forall x' \in \mathbf{X}', \forall v \in V, v \notin M(x'), \exists w \in N(v) : d_X(id(w), x) < d_X(id(v), x)$. For a graph $G = (V, E)$ with such an embedding, the tuple (G, id, C, ca) is called a content addressable storage.

Definition III.2 guarantees that greedy routing terminates at the closest node to an address x . Thus, nodes can store and retrieve files using greedy routing. However, Definition III.2 does not demand that the content is distributed on the nodes in a balanced manner. Thus, we now characterize the notion of balanced or fair content addressing.

Definition III.3. Let (G, id, C, ca) be a content addressable storage. Furthermore, $\forall v \in V$ let $\mathbf{B}(v) = \{x \in \mathbf{X}' : \forall w \in V d_X(id(v), x) \leq d_X(id(w), x)\}$ be the set of coordinates in \mathbf{X}' closest to v , and μ be the normalized point measure, i.e., μ maps a subset E of \mathbf{X}' to the fraction of coordinates contained in E . (G, id, C, ca) is said to be (f, δ) -balanced for a real-valued factor $f \geq 1$ if

$$\forall v \in V, \mu(\mathbf{B}(v)) \leq f \cdot \frac{1}{|V|} + \delta. \quad (1)$$

An embedding algorithm \mathbf{A} is called (f, δ) -balanced if it generates embeddings id such that the content addressable storage (G, id, C, ca) is (f, δ) -balanced.

Essentially, Definition III.3 states that the expected fraction of content assigned to a node should at most be f times the average content assigned to each node. A well-known example for balanced content addressing on freely adaptable topologies are DHTs. In DHTs, file addresses correspond to b -bit hashes of either the file's name, description, or content. DHTs are $(\mathcal{O}(\log n), 0)$ -balanced [10].

We now shortly motivate some details in Definition III.3. The additional term δ is assumed to be small in comparison to $1/n$. Its purpose is mainly to compensate for rounding errors emerging from the fact that, for a finite X , n most likely does not evenly divide $|X|$. The use of the normalized point measure μ is only sensible if the file addresses $ca(C)$ are approximately uniformly distributed. Otherwise, $\mu(A)$ should correspond to the measure of the preimage $ca^{-1}(A)$. However, on the one hand, the latter definition requires an introduction to measure theory. On the other hand, we would need to express the difference between a pseudo-random hash function, which we use for constructing the addressing scheme ca , and a uniform distribution in terms of the parameter δ , which is out-of-scope for this paper. Thus, we restrict our goals to balancing the fraction of content addresses mapped to a node rather than the fraction of content mapped to the node.

C. Dynamics and Stabilization

Now, we assume that the topology of the graph changes over time. Here, each topology change refers to the addition and removals of *one* node or edge. We model the graph topology over time as a stochastic process $(G_t)_{t \in \mathbb{N}}$ such that

$G_t = (V_t, E_t)$ represents the graph after the t -th topology change. When the topology changes, the embedding has to be adapted, so that we have a time-dependent embedding id_t . In contrast, we assume that the set of potential content C and the addressing function ca remain unchanged. In order for the content addressable storage (G_t, id_t, C, ca) to continuously function effectively for all t , the embedding has to be adjusted to maintain greedy and balanced. We now define two properties for an embedding algorithm, before formally defining the concept of a content addressable storage in a dynamic scenario.

Definition III.4. Let \mathbf{A} be an embedding algorithm for content addressable greedy embeddings id based on spanning trees $T = (V, E^T)$, $E^T \subseteq E$. We write $\mathbf{A}(T, \emptyset)$ to indicate that \mathbf{A} is applied on the tree T . We call \mathbf{A} dynamic if we can compute $\mathbf{A}(T_u, id(u))$ on a subtree $T_u = (V_u, E_u^T)$ with $V_u \subseteq V$, $E_u^T \subseteq E^T$ rooted at a node u such that

- 1) $\mathbf{A}(T_u, id(u))$ only changes coordinates of nodes $v \in V_u$,
- 2) the communication complexity of $\mathbf{A}(T_u, id(u))$ is $\mathcal{O}(|V_u|)$, and
- 3) for any tree $T'_u = (V'_u, E'_u)$ rooted at u , graph $G' = (V', E')$ with $V' = V \setminus V_u \cup V'_u$ and spanning tree $T' = (V', E^{T'})$ with $E^{T'} = E^T \setminus E_u^T \cup E'_u$, $\mathbf{A}(T'_u, id(u))$ results in an embedding id' such that

$$\forall x \in \mathbf{X}', M(x) \in V_u \implies M'(x) \in V'_u.$$

Furthermore, \mathbf{A} is called dynamic (f', δ) -balanced if it is dynamic and

$$\forall v \in V_u : \mu(\mathbf{B}(v)) \leq \left(\sum_{v_0 \in V_u} \mu(\mathbf{B}(v_0)) \right) \frac{f'}{|V_u|} + \delta. \quad (2)$$

holds for any embedding generated by $\mathbf{A}(T_u, id(u))$.

In other words, Definition III.4 requires an embedding algorithm to be able to re-embed local subtrees with changed nodes and edges such that the local embedding is balanced, covers the addresses of the previous embedding, and other nodes and their content addresses are unaffected. Note that this local balance does not imply global balance. If the combined fraction of coordinates $M_0 = \sum_{v_0 \in V_u} \mu(\mathbf{B}(v_0))$ mapped to the nodes in the subtree is disproportionately high in comparison to the number of nodes in the subtree, the fraction of coordinates mapped to each node in the subtree might exceed f/n . A stabilization algorithm should thus decide if the embedding algorithm \mathbf{A} can be applied locally or if the re-embedding has to consider additional nodes in order to balance the storage responsibilities.

Definition III.5. Let \mathbf{A} be a (f', δ) -balanced embedding algorithm with $f' \leq f$. A stochastic process $((G_t, id_t)_{t \in \mathbb{N}}, C, ca, \mathbf{S}(\mathbf{A}))$ is called a dynamic (f, δ) -balanced content addressable storage if the stabilization algorithm $\mathbf{S}(\mathbf{A})$ ensures that (G_t, id_t, C, ca) is a (f, δ) -balanced content addressable storage for all $t \in \mathbb{N}$.

Definition III.5 allows the stabilization algorithm to be parameterized by the embedding algorithm. In this manner,

we allow for a general stabilization algorithm that calls upon a variable dynamic embedding algorithm.

IV. ALGORITHM DESIGN

In this section, we develop an efficient stabilization algorithm that can restore a $(\mathcal{O}(D), \delta)$ -balanced content addressable storage after a topology change with D denoting an upper bound on the spanning tree depth. We first consider the algorithm design from a high-level point of view. More precisely, we show that we can construct such a stabilization algorithm $\mathbf{S}(\mathbf{A})$ on the basis of any dynamic $(1, \delta)$ -balanced embedding algorithm \mathbf{A} . We then present a concrete algorithm \mathbf{A} for our evaluation. Last, we introduce potential variations and improvements of our algorithm for practical use.

A. Stabilization

The key idea of algorithm $\mathbf{S}(\mathbf{A})$ is that a node u can locally decide if it re-embeds its subtree or forwards a request for re-embedding to its parent. Throughout this section, let $T_u = (V_u, E_u)$ denote the subtree rooted at u . In order to decide if a local re-embedding is possible, u maintains an estimate $n_{est} \in [n/g, ng]$ of the number of nodes n in the network. Furthermore, u keeps track of its number of descendants $|V_u|$ as well as the combined fraction $cont(V_u) = \sum_{v \in V_u} \mu(\mathbf{B}(v))$ of addresses assigned to nodes in V_u . Similarly, for each child v , u keeps track of $|V_v|$. We first describe the idea of how the dynamic re-embedding in the presence of topology changes works. Then, we detail how to obtain the required knowledge for making the local decision to re-embed. Last, we present the pseudocode of the stabilization algorithm.

a) *Maintaining Stability*: We aim to maintain a (f, δ) -balanced content addressable storage with $f = \mathcal{O}(D)$ in the presence of topology changes. We assume that there exists a $(1, \delta)$ -balanced embedding algorithm \mathbf{A} . The topology change and subsequent spanning tree stabilization either replaces a subtree T_u with a subtree $T'_u = (V'_u, E'_u)$ or construct a new spanning tree. We focus on the former case as the latter requires re-embedding the complete graph. If u re-embeds locally, i.e., applies the algorithm \mathbf{A} only to T'_u , we have $cont(V'_u) = cont(V_u)$ by the third condition in Definition III.4. Then Eq. 2 states that the maximal fraction of addresses assigned to any node v in V'_u is

$$\mu(\mathbf{B}(v)) \leq \frac{cont(V_u)}{|V'_u|} + \delta, \quad (3)$$

because \mathbf{A} is $(1, \delta)$ -balanced. We can express Eq. 3 in the form $\frac{f'}{n} + \delta$ with $f' = n \cdot cont(V_u)/|V'_u|$. If indeed $n_{est} \in [n/g, ng]$ for a global parameter g , we have $n \leq n_{est}g$ and hence $f' \leq n_{est}g \frac{cont(V_u)}{|V'_u|}$. Thus, if for some $f(u) \leq f$

$$n_{est}g \frac{cont(V_u)}{|V'_u|} \leq f(u), \quad (4)$$

re-embedding locally guarantees that $\mu(\mathbf{B}(v)) \leq \frac{f}{|V|} + \delta$ for all $v \in V'_u$, so that we indeed maintain a (f, δ) -balanced content addressable storage. If Eq. 4 does not hold, u cannot guarantee that local re-embedding maintains a $(f(u), \delta)$ -balanced content

addressable storage. Then u contacts its parent $p(u)$ with a request for re-embedding. The node $p(u)$ decides if it should re-embed locally, changing the coordinates within subtrees rooted at u and its siblings, or if it relays the request to its own parent. In this manner, nodes might forward the request for re-embedding to the root who can always re-embed such that the resulting content addressable storage is $(1, \delta)$ - and hence (f, δ) -balanced.

It remains to consider how to choose $f(u)$. As stated above, we need to ensure that $f(u) \leq f$. On the first glance, the choice $f(u) = f$ seems suitable as it maximizes the probability that Eq. 4 holds. However, if indeed $f = f(u)$, the re-embedding might only barely restore the desired balance. Any further change affecting any of the subtrees might thus lead to an immediate need for another re-embedding. In order to allow to reduce the frequency of the re-embedding, we thus choose a level-dependent $f(u)$. More precisely, a parent v provides an embedding with a lower balance factor than the child node u , i.e., $f(v) < f(u)$. In this manner, the probability that u has to contact its parent for a further re-embedding shortly after such an re-embedding decreases. In order to maintain an overall balance factor $f = \mathcal{O}(D)$, we choose the local balance factor corresponding to the level of the node in the tree, i.e.,

$$f(u) = g(1 + c + level(u)). \quad (5)$$

Using the size approximation accuracy g as factor ensures that a re-embedding is not only necessary due to the uncertainty about the current global state despite a good balance in the subtree. The *tree depth offset* c allows a trade-off between the accepted level of imbalance and the stabilization overhead. So, an increased parameter c implies that the maximal fraction of content per node can be high but might reduce the frequency of coordinate changes.

b) *Updating State Information*: The estimate n_{est} as well as the quantities $|V_v|$ and $cont(V_u)$ are essential to check if Eq. 4 holds. The fraction $cont(V_u)$ depends on the nature of the coordinate space \mathbf{X} and the address space $\mathbf{X}' \subset \mathbf{X}$. Thus, computing them depends on the nature of the embedding algorithm \mathbf{A} and the addressing scheme ca . Here, we thus only describe how to obtain $cont(V_u)$ during the design of \mathbf{A} . Here, we focus on maintaining the network size estimate n_{est} . In the process, we also obtain and maintain the subtree sizes $|V_v|$. Note that $n = |V_r|$ for the root r . Upon initialization, we derive the network size $n = |V| = |V_r|$ recursively. Each node v forwards the size V_v to its parent, starting at leaves, which send $|V_v| = 1$. As soon as a node u has received $|V_v|$ from all its children v , u sends $1 + \sum_{v \in children(u)} |V_v|$ to its parent. Finally, r obtains the current network size and broadcasts it to all nodes along the edges of the tree. Later on, whenever a node u accepts an additional child or one of its children departs, u sends the new value of $|V_u|$ to its parent. All subtree sizes along the path to the root are subsequently updated. After the root node has updated its local state, it checks if the current value for $|V_r| = n$ and the global estimate n_{est} still satisfy

$n_{est} \in [n/g, ng]$. If not, r broadcasts the new estimate and at the same time runs the re-embedding algorithm.

Algorithm 1 $S(\mathbf{A})(u, v, |V'_v|, b)$

```

1: #  $u$ : node,  $v$ : child of  $u$ ;  $|V'_v|$ : updated  $|V_v|$ ,  $b$ : forward flag
2: # Global: balance factors  $f, g, c$ ; size estimate  $n_{est}$ ; Alg.  $\mathbf{A}$ 
3: # State  $u$ : content  $cont(V_u)$ ; subtree sizes  $|V_v|$ ; parent  $p(u)$ 
4:  $|V_v| = |V'_v|$ 
5:  $|V_u| = 1 + \sum_{v \in children(u)} |V_v|$ 
6: if  $u$  is root then
7:   if  $|V_u| < n_{est}/g$  or  $|V_u| > n_{est}g$  or not  $b$  then
8:      $n_{est} = |V_u|$ 
9:      $\mathbf{A}(u)$ ,
10:    Broadcast  $n_{est}$ 
11:   end if
12: else if  $b$  then
13:    $S(\mathbf{A})(p(u), u, |V_u|, b)$ 
14: else
15:   if  $n_{est}g \frac{cont(V_u)}{|V_u|} \leq g(1 + c + level(u))$  then
16:      $\mathbf{A}(u)$ 
17:      $S(\mathbf{A})(p(u), u, |V_u|, true)$ 
18:   else
19:      $S(\mathbf{A})(p(u), u, |V_u|, false)$ 
20:   end if
21: end if

```

c) Pseudocode: Algorithm 1 displays the pseudo code governing a node u 's reaction to a topology change in T_u . The algorithm combines the decision for re-embedding with updates of local state information. The input of the algorithm is the current node u , the child v that is affected by the change, the new value for $|V_v|$, and a flag b indicating that the re-embedding has already been taken care of. Hence, if b is true, u only has to forward the updated subtree sizes to the root. The system parameters are the balance factor f and the estimation quality g . In addition, each node stores the same global network size estimate n_{est} . The local state at u includes the fraction of content $cont(V_u)$, the number of nodes in subtrees rooted at its children, and the parent $p(u)$. In Lines 4 and 5, u updates its information regarding the subtree sizes. Lines 6-10 specify the behavior of the root. Note that the root can always generate an (f, δ) -balanced content addressable storage, so that checking Eq. 4 are not necessary. Rather, the root calculates a new estimate n_{est} and re-embeds the graph whenever the old estimate is not accurate enough or its descendants have been unable to locally re-embed, i.e., if b is false. If u is not the root of the tree, u first checks the flag b . If b is true, u merely relays the updated subtree sizes to the parent (Line 13). Otherwise, u has to decide if it locally re-embeds or relays the request for re-embedding to its parent. The decision in Line 15 follows Eq. 4. If the local state information satisfies Eq. 4, u executes the embedding algorithm on T_u and forwards the updated subtree sizes to its parent. Furthermore, u sets the flag b to true (Lines 16 and 17). If u cannot maintain the necessary balance by locally re-embedding, u forwards the updated subtree sizes to the parent together with b set to false, indicating the need for a re-embedding (Line 19). In this manner, Algorithm 1 recursively

restores a (f, δ) -balanced content addressable storage.

We consider potential variations and speedups for the use of $S(\mathbf{A})$ in practice in Section IV-C.

B. Embedding

Our embedding algorithm is a modification of the unbalanced content addressing scheme for Prefix Embedding [8]. In a nutshell, the idea of our algorithm is to count each node in Prefix Embedding as multiple nodes.

Prefix Embedding, a variation of PIE [6], encodes the position of a node with regard to the root of the tree. More precisely, each node u enumerates the edges to its children. The coordinate then corresponds to the vector of edge numbers on the unique path from the root to the respective node. The distance between two such coordinates corresponds to the sum of the length of their coordinate vectors, subtracting twice the number of leading equal elements (the common prefix). In this manner, the distance between two node coordinates equals the length of the path between the two nodes in the spanning tree.

We modify Prefix Embedding by replacing the numerical elements of the vectors with sets of integers in an interval. The length of each interval depends on the number of nodes in the corresponding subtree. So, a node on level l divides the space of 2^b numbers for the $l+1$ -th element of the coordinate vector evenly between itself and its descendants, as displayed in Algorithm 2. More precisely, a node u receives a prefix for all nodes in V_u from its parent, starting with an empty prefix at the root. The prefix corresponds to u 's own coordinate $id(u)$ and consists of l intervals. After receiving its coordinate, u assigns coordinates consisting of $l+1$ intervals to its children. For the i -th child v_i , u adds the set of integers in the interval $\left[\sum_{j=1}^{i-1} \frac{|V_{v_j}|}{|V_u|} 2^b, \sum_{j=1}^i \frac{|V_{v_j}|}{|V_u|} 2^b \right]$, which has cardinality of at most $\lceil \frac{|V_{v_i}|}{|V_u|} 2^b \rceil$ (Lines 5-9). The subtree rooted at the child is then recursively embedded.

We now formally derive the coordinate space and the distance function for the assigned coordinates. First, we denote the set of all integers within an interval $[z_1, z_2]$ by $Ic(z_1, z_2) = \{i : i \in [z_1, z_2], i \in \mathbb{Z}\}$. Furthermore, let $IC = \{Ic(z_1, z_2) : z_1, z_2 \in [0, 2^b], z_2 \geq z_1\}$ denote the set of all such sets with $0 \leq z_1 \leq z_2 < 2^b$. Then, our coordinate space $\mathbf{X} = IC^*$ corresponds to all vectors with entries in IC . The distance between two node coordinates is analogous to Prefix Embedding: the difference of the sum of the coordinates lengths and twice the common prefix length. However, in order to allow for balanced content addressing, we use a slightly different concept than the common prefix length to compare vector elements. Rather than only considering equal elements, we consider two sets a match if one is contained in the other. Formally, let I_1 and I_2 denote two sets of integers. Then we set $sub(I_1, I_2) = true$ if $I_1 \subseteq I_2$ or $I_2 \subseteq I_1$. As a consequence, we denote the *contained interval length* of two vectors $x_1, x_2 \in \mathbf{X}$ as $cil(x_1, x_2) = \max\{j \in \{0, \dots, \min\{D(x_1), D(x_2)\}\} : sub(x_1(j), x_2(j))\}$. Hence, the distance between two coordinates is

$$d_X(x_1, x_2) = D(x_1) + D(x_2) - 2 \cdot cil(x_1, x_2). \quad (6)$$

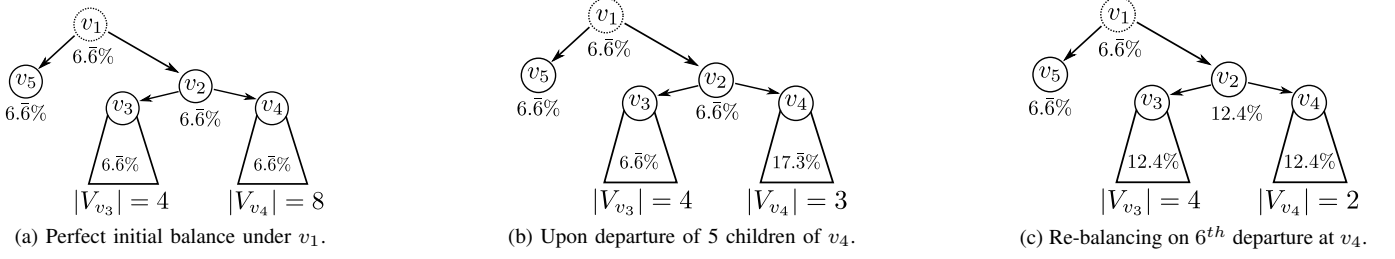


Fig. 1: $\mathbf{S}(\mathbf{A})$ on 15 nodes (triangles below v_3 and v_4 denote branches of $|V_{v_3}|$ and $|V_{v_4}|$ nodes), $g = 2$ and $c = 0$: Arrows denote parent child relationships. Percentages given for branches denote the fraction allocated to each node in the branch, percentage given for top nodes the fraction allocated to the respective node; Fig. 1a: the content addressing initially allocates $1/15^{th}$ to each node, achieving perfect balance. Departure of 5 children under v_4 is balanced by v_4 (Fig. 1b), allocations increase to 17.3% per node. The sixth departure triggers escalation and the re-embedding request is relayed to v_2 (Fig. 1c).

with $D(x)$ denoting the dimension of a vector x .

Next, we consider the file address generation ca for files $c \in C$. Typically, ca corresponds to a hash function $h : C \rightarrow H = \mathbb{Z}_{2^b}$. However, as our coordinates are vectors, we choose the address space $\mathbf{X}' = \{\{a\} : a \in \mathbb{Z}_{2^b}\}^L$ corresponding to vectors of a fixed length L with $L \geq D$ exceeding the spanning tree depth. We use multiple salted hashes to obtain the address of a file c , i.e., the i -th element of $ca(c)$ is $y = (\{y_1\}, \dots, \{y_L\})$ for $y_i = h(c + i)$. The file c is then stored at the node with the closest coordinate $id(u)$ to $ca(c)$ according to Eq. 6. The node u can be located using greedy routing by forwarding a request to store or retrieve $ca(c)$ to the closest neighbor until no such neighbor exists. If the coordinate of u changes due to the dynamics, u needs to start a new storage request for c to ensure that the file is indeed stored on the closest node. We now show that \mathbf{A} is greedy content addressable, $(1, \delta)$ -balanced, and hence any (G, id, C, ca) is a content addressable storage.

Algorithm 2 $\mathbf{A}(T_u, id(u))$

```

1: #  $T_u$ : subtree to embed,  $Ic(z_1, z_2)$ : integers in  $[z_1, z_2]$ 
2: #  $b$ : length of coordinate elements,  $||$ : concatenation
3:  $ol = 0$ 
4:  $next = 0$ 
5: for  $v \in children(u)$  do
6:    $next = next + |V_v|$ 
7:    $id(v) = id(u) || Ic(\frac{ol}{|V_u|} 2^b, \frac{next}{|V_u|} 2^b)$ 
8:    $\mathbf{A}(T_v, id(v))$ 
9:    $ol = next$ 
10: end for

```

Proposition IV.1. *The dynamic embedding algorithm \mathbf{A} is content addressable greedy. If $L \leq b$ with L being the upper bound on the tree depth, then \mathbf{A} is $(1, \frac{L+1}{2^b})$ -balanced.*

Proof:

In order to show that \mathbf{A} is content addressable greedy, we leverage the corresponding results for Prefix Embedding. In the main part of the proof, we show that \mathbf{A} is $(1, \frac{L+1}{2^b})$ -balanced by determining f and δ as in Definition III.3. For this purpose, we first determine an upper bound for $\mathbf{B}(v)$ based

on Eq. 6 and then leverage this upper bound to confirm that $\delta \leq \frac{L+1}{2^b}$ for $f = 1$.

At first, we show that the coordinate spaces \mathbf{X}, \mathbf{X}' as well as the distance d_X defined in Eq. 6 can be mapped to a Prefix Embedding. Before showing that \mathbf{A} is content addressable greedy, we present an alternative interpretation of the coordinate spaces \mathbf{X} and \mathbf{X}' and the distance d_X defined in Eq. 6. It is easy to see that \mathbf{X}' and $\mathbb{Z}_{2^b}^L$ are equivalent: we map a vector of singletons $x' = (\{a_1\}, \dots, \{a_L\})$ to a vector $\tilde{x} = (a_1, \dots, a_L)$. Similarly, we can associate \mathbf{X} with a subset Y of $\mathcal{P}(\mathbb{Z}_{2^b}^L)$ with \mathcal{P} indicating the power set, i.e., the set of all subsets. Here, we map a vector $x = (Ic(x_1(1), x_1(2)), \dots, Ic(x_L(1), x_L(2)))$ to the set $\tilde{x} = \{(y_1, \dots, y_L) : y_i \in Ic(x_i(1), x_i(2))\}$. Now, $\{a_i\} \subseteq Ic(x_i(1), x_i(2))$ holds iff there exists $y \in \tilde{x}$ with $y_i = a_i$. For $c = \max_{y \in \tilde{x}} cpl(y, \tilde{x}')$ and by the definition of $cil(x, x')$, we have $cil(x, x') \leq c$ and $cil(x, x') \geq c$, so that $cil(x, x') = \max_{y \in \tilde{x}} cpl(y, \tilde{x}')$.

Leveraging the alternative definition of cil , we show that \mathbf{A} is content addressable greedy by relating \mathbf{A} to Prefix Embedding. Consider a node u with coordinate $id(u) = (Ic(x_1^u(1), x_1^u(2)), \dots, Ic(x_L^u(1), x_L^u(2)))$. Replace u with a set of nodes $ID(u)$ of size $\prod_{i=1}^L |Ic(x_i^u(1), x_i^u(2))|$ and assign each node $u' \in ID(u)$ a unique coordinate $id_{PRE}(u') = (y_1, \dots, y_L)$ with $y_i \in Ic(x_i^u(1), x_i^u(2))$. For every neighbor v of u , connect u' with all $v' \in ID(v)$. In particular, we have an edge between u' and all $v' \in ID(v)$ such that $id_{PRE}(v') = (y_1, \dots, y_L, z)$ for some integer z . Thus, the resulting embedding id_{PRE} is an instance of Prefix Embedding and hence content addressable greedy as shown in [8]. As a consequence, greedy routing for an address $\tilde{x}' \in \mathbb{Z}_{2^b}^L$ traverses a path (v'_0, \dots, v'_t) such that $id_{PRE}(v'_t)$ is closest to \tilde{x}' in terms of $d_{PRE}(\tilde{x}', id_{PRE}(v'_t)) = D(\tilde{x}') + D(id_{PRE}(v'_t)) - 2cpl(\tilde{x}', id_{PRE}(v'_t))$. For the equivalent address $x' \in \mathbf{X}'$, we have $cil(x', id(v)) = \max_{v' \in ID(v)} cpl(\tilde{x}', id_{PRE}(v'))$ and $d_X(x', id(v)) = \min_{v' \in ID(v)} d_{PRE}(\tilde{x}', id_{PRE}(v'))$. Hence, greedy routing for an address in the embedding id traverses the path (v_0, \dots, v_t) and terminates at the node with the closest coordinate to x' . Thus, \mathbf{A} is content addressable greedy.

Now, we show that \mathbf{A} is $(1, \frac{L}{2^b})$ -balanced. First, we derive the cardinality $|\mathbf{B}(v)| = \mu(\mathbf{B}(v)) \cdot (2^b)^L$. Let $x = id(v)$. By

the definition of the distance d_X in Eq. 6, $\mathbf{B}(v)$ consists of all vectors $x' \in X'$ such that i) $cil(x, x') = D(x)$, and ii) there is no child u of v with $cil(id(u), x') > D(x)$. We extend x to a coordinate x_L of length L such that $\mathbf{B}(v) = \{x' \in \mathbf{X}' : d_X(x', x_L) = 0\}$ by adding elements $Ic(x_i^v(1), x_i^v(2))$ for $i > D(x)$. With

$$z_m(v) = \begin{cases} 0, & \text{children}(v) = \emptyset \\ \max_{w \in \text{children}(v)} x_{D(x)+1}^w(2), & \text{otherwise} \end{cases}$$

denoting the smallest integer such that $z_m \notin Ic(x_{D(x)+1}^w(1), x_{D(x)+1}^w(2))$ for any child w of v , we set $Ic(x_{D(x)+1}^v(1), x_{D(x)+1}^v(2)) = Ic(z_m(v), 2^b)$. Furthermore, for $i > D(x) + 1$, we set $Ic(x_i^v(1), x_i^v(2)) = Ic(0, 2^b)$ to cover all possible elements. On the one hand, $x' = (\{a_1\}, \dots, \{a_L\}) \in \mathbf{B}(v)$ implies $a_i \in Ic(x_i^v(1), x_i^v(2))$ for all i and hence $d_X(x_L, x') = 0$. On the other hand, $x' \notin \mathbf{B}(v)$ implies $a_i \notin Ic(x_i^v(1), x_i^v(2))$ for some $i \leq D(x) + 1$ and hence $d_X(x_L, x') \neq 0$. So, indeed $\mathbf{B}(v) = \{x' \in \mathbf{X}' : d_X(x', x_L) = 0\}$ and hence

$$\begin{aligned} |\mathbf{B}(v)| &= \prod_{i=1}^L |Ic(x_i^v(1), x_i^v(2))| \\ &= 2^{b(L-D(x)-1)} \prod_{i=1}^{D(x)+1} |Ic(x_i^v(1), x_i^v(2))|. \end{aligned} \quad (7)$$

In the following, let v_l denote the ancestor of v on level l . In particular, $v = v_{D(x)}$. As stated in Line 7 of Algorithm 2, we have $Ic(x_i^v(1), x_i^v(2)) = Ic(\lfloor \frac{r}{|V_{v_{i-1}}|} \rfloor 2^b, \lceil \frac{r+|V_{v_i}|}{|V_{v_{i-1}}|} \rceil 2^b)$ for $i \leq D(x)$ with some $r \in [0, |V_{v_{i-1}}| - |V_{v_i}|]$. This implies an upper bound $|Ic(x_i^v(1), x_i^v(2))| \leq \lceil \frac{|V_{v_i}|}{|V_{v_{i-1}}|} \rceil 2^b + 1$. For $i = D(x) + 1$, the number of integers not assigned to any of the children is bound by $\frac{1}{|V_v|} 2^b + 1$. Inserting these bounds into Eq. 7 yields

$$|\mathbf{B}(v)| \leq 2^{b(L-D(x)-1)} \left(\frac{1}{|V_v|} 2^b + 1 \right) \prod_{i=1}^{D(x)} \left(\frac{|V_{v_i}|}{|V_{v_{i-1}}|} 2^b + 1 \right). \quad (8)$$

In the second step of the proof, we derive δ from Eq. 8. For this purpose, we write

$$\prod_{i=1}^{D(x)} \left(\frac{|V_{v_i}|}{|V_{v_{i-1}}|} 2^b + 1 \right) = \sum_{i=0}^{D(x)} c_i (2^b)^i \quad (9)$$

with

$$c_{D(x)} = \prod_{i=1}^{D(x)} \frac{|V_{v_i}|}{|V_{v_{i-1}}|} = \frac{|V_{v_{D(x)}}|}{|V_{v_0}|} = \frac{|V_v|}{n}.$$

As $\frac{|V_{v_i}|}{|V_{v_{i-1}}|} \leq 1$, we have $\prod_{i=1}^{D(x)} \left(\frac{|V_{v_i}|}{|V_{v_{i-1}}|} 2^b + 1 \right) \leq (2^b + 1)^{D(x)} = \sum_{i=0}^{D(x)} \binom{D(x)}{i} (2^b)^i$ and $c_i \leq \binom{D(x)}{i}$. Consequently, we obtain upper bounds $c_{D(x)} \leq D(x) \leq L$ and

for $D(x) > 1$

$$\begin{aligned} \sum_{i=0}^{D(x)-2} c_i (2^b)^i &\leq (2^b)^{D(x)-2} \sum_{i=0}^{D(x)-2} \binom{D(x)}{i} \\ &\leq (2^b)^{D(x)-2} \sum_{i=0}^{D(x)} \binom{D(x)}{i} = (2^b)^{D(x)-2} 2^{D(x)} \leq (2^b)^{D(x)-1} \end{aligned}$$

The last step uses $2^b \geq 2^L \geq 2^{D(x)}$. Inserting Eq. 9 and the upper bounds on the coefficients c_i in Eq. 8, we obtain

$$|\mathbf{B}(v)| \leq \frac{1}{n} (2^b)^L + L (2^b)^{L-1} + (2^b)^{L-1}.$$

Division by the number of addresses $|X'| = (2^b)^L$ shows that \mathbf{A} is indeed $(1, \frac{L+1}{2^b})$ -balanced. ■

C. Improvements and Variations

There are multiple possibilities to slightly reduce the overhead of $\mathbf{S}(\mathbf{A})$ in practice or achieve additional properties.

Delay before broadcasting new estimate: In Line 10 of Algorithm 1, the root broadcasts a new estimate immediately after receiving an update. However, a node or edge departure might result in a temporarily low estimate as the descendants of the departed nodes select alternative parents. Adding a short delay before reacting to a considerable change in the network size avoids broadcasting a new estimate without the actual need to do so. During our theoretical and practical evaluation, we assume that the root waits until its size estimate is accurate.

No local re-embedding after joins: In Algorithm 1, the parent u of a newly joined node v re-embeds the complete subtree V_v . However, as long as the load is sufficiently balanced in the subtree, such an action might unnecessarily increase the overhead. Rather, u can assign v a preliminary coordinate in $\mathbf{B}(u)$ by adding $Ic(max, (max + 2^b)/2)$ to v 's coordinate with max denoting the highest number assigned to a last coordinate of u 's children. In this manner, we postpone the re-embedding of the subtree until one of its children leaves or asks for a re-embedding.

Virtual binary trees: In addition to reducing the frequency of new coordinate assignments, the number of nodes affected by a re-embedding can be reduced by only changing the coordinates within a subset of the subtrees rooted at children. For this purpose, we leverage the concept of virtual binary trees presented in [8]. Here, we represent a subgraph consisting of a parent and its children as a binary tree such that the children are the leaves and the parent executes the functionality of all internal nodes. In this manner, if a node u receives a re-embedding request relayed from one of its children, u first checks if it can balance a set of two or three subtrees. If re-embedding only those trees is possible according to Algorithm 1, the remaining subtrees remain unaffected. Otherwise, nodes subsequently considers subtrees at a lower level of the virtual binary tree until it can either locally re-embed or has to relay the request to its own parent. As the structure of the virtual subtree rooted at a node changes

whenever the number of children changes, the successive consideration only applied for nodes other than the parent of the joined or departed node. Note that the additional nodes of the virtual binary trees do not count into the network size but the levels considered in Line 15 of Algorithm 1 correspond to the levels in the virtual tree in order to avoid short-lived re-embeddings.

Estimated subtree sizes: Algorithm 1 relies on the actual sizes of subtrees. In particular, leaves reveal that they have no descendants. Revealing such topology information is potentially undesired in privacy-preserving communication systems such as F2F overlays and might reduce the anonymity. Hence, the subtree size can be obfuscated. For instance, rather than adding 1 for each node, each node can be counted as either 0, 1, or 2. We obtain an unbiased estimate as long as the probabilities for 0 and 2 are equal. In order to avoid inferences over time, each node's count should be consistent. In our practical evaluation, we consider the impact of using such estimates.

Heterogeneous node resources: If the storage of nodes differs considerable, \mathbf{A} and $\mathbf{S}(\mathbf{A})$ should consider such heterogeneous resources. We extend the above idea to not necessarily count each node with 1. Instead, the count of a node and hence the expected assigned content should correspond to a node's resources. In other words, we replace the subtree sizes in Algorithms 1 and 2 with the overall storage capacity of the nodes in the subtrees. A detailed evaluation of the heterogeneous node resources is out-of-scope for this paper due to the lack of realistic models but should be considered in greater detail in the future.

V. ANALYSIS

We designed a stabilization algorithm $\mathbf{S}(\mathbf{A})$ for a dynamic $(\mathcal{O}(D), \delta)$ -balanced content addressable storage. Now, we derive the communication complexity of $\mathbf{S}(\mathbf{A})$. We start by showing an essential Lemma concerning the expected number of descendants. We subsequently treat node joins and node departures separately. Throughout the section, we assume that the depth of the tree is at most D and we write $\mathbb{E}(X)$ for the expected value of some described random variable X . In particular, we write $\mathbb{E}(S)$ for the expected number of siblings of a random node. We focus on the ideas of the proofs. The complete proofs can be found in our technical report [17].

Lemma V.1. *The expected number of descendants of a random node is $\mathbb{E}(Y) = \mathcal{O}(D)$.*

Proof: The idea of the proof is to make use of the fact that the average number of descendants is equal to the average number of ancestors. The latter corresponds to the average level of a node and is hence bound by the depth of tree. Formally, let $Z = \{(u, v) : u \text{ is descendant of } v\}$ denote the set of all descendant-ancestor relations. The expected number of descendants is hence given by $|Z|/n$. We now determine an upper bound on $|Z|$. For this purpose, let L_u denote the level of a node u . A node on level l of the spanning tree is a

descendant of l nodes, so that the total number of descendant-ancestor relations corresponds to the sum of all levels. Hence

$$|Z| = \sum_{u \in V} L_u \leq n|D|.$$

Division by n shows the claim. \blacksquare

Proposition V.2. *The communication complexity of Algorithm 1 for a node join is $\mathbb{E}(\text{cost}_{\text{join}}(\mathbf{S}(\mathbf{A}))) = \mathcal{O}(D)$*

Proof: We write the communication complexity as a sum of three phases. First, X_1 denotes the complexity of local re-embeddings, corresponding to Line 16 in Algorithm 1. Second, X_2 denotes the complexity of propagating status updates to the root (Lines 13, 17 and 19). Third, X_3 denotes the communication complexity after the updates have reached the root (Lines 6-10). So, $\mathbb{E}(\text{cost}_{\text{join}}(\mathbf{S}(\mathbf{A}))) = \mathbb{E}(X_1) + \mathbb{E}(X_2) + \mathbb{E}(X_3)$.

We start by considering X_1 . Note that the parent node u of the newly joined node v calls \mathbf{A} rather than relaying the request as the quantity on the left of Eq. 4 is reduced and thus remains smaller than $f(u)$. Thus, X_1 corresponds to the complexity of applying \mathbf{A} to a subtree consisting of u , v , and all of Y descendants of u . By the third condition in Definition III.4 and Lemma V.1, $\mathbb{E}(X_1) = \mathcal{O}(2 + \mathbb{E}(Y)) = \mathcal{O}(D)$ follows.

The number of propagated updates X_2 is bound by the longest path from a node to the root, hence $\mathbb{E}(X_2) = \mathcal{O}(D)$.

In order to determine X_3 , let E denote the event that the new network size n after the join exceeds $n_{\text{est}g}$. We have $\mathbb{E}(X_3) = \mathbb{E}(X_3|E)P(E)$, because the root only sends additional messages if a new estimate needs to be broadcast. The complexity of re-embedding and broadcasting n_{est} is $\mathbb{E}(X_3|E) = \mathcal{O}(n)$. However, the event E implies that $n = n_{\text{est}g} + 1$, hence it only occurs after at least $n_{\text{est}}(g - 1) + 1$ nodes joined. Hence E occurs only for a fraction $P(E) = \mathcal{O}(\frac{1}{n})$ of the joins. So, $\mathbb{E}(X_3) = \mathcal{O}(1)$ and indeed

$$\mathbb{E}(\text{cost}_{\text{join}}(\mathbf{S}(\mathbf{A}))) = \mathbb{E}(X_1) + \mathbb{E}(X_2) + \mathbb{E}(X_3) = \mathcal{O}(D).$$

\blacksquare

Proposition V.3. *The communication complexity of Algorithm 1 for a node departure is $\mathbb{E}(\text{cost}_{\text{depart}}(\mathbf{S}(\mathbf{A}))) = \mathcal{O}(D^3 \mathbb{E}(S))$.*

Proof: Analogously to the proof of Proposition V.2, we derive the desired bound as the sum of four phases X_1 , X_2 , X_3 , and X_4 . The decisive quantity is $\mathbb{E}(X_1)$, which corresponds to local re-embeddings by an ancestor of the departed node as a direct reaction to the departure rather than to a re-join of a descendant of the departed node. As in the proof of Lemma V.2, X_2 and X_3 denote the complexity of propagating updates to and from the root with $\mathbb{E}(X_2) = \mathcal{O}(D)$ and $\mathbb{E}(X_3) = \mathcal{O}(1)$. X_4 denotes the complexity resulting from the re-joins of the descendants of the departed node. By Lemma V.1 and Lemma V.2, these correspond to an expected number of $\mathcal{O}(D)$ joins at an expected communication complexity of $\mathcal{O}(D)$ each, hence $\mathbb{E}(X_4) = \mathcal{O}(D^2)$.

The main difficulty lies in deriving $\mathbb{E}(X_1)$. We obtain a global upper bound on the probability p that a node v has to

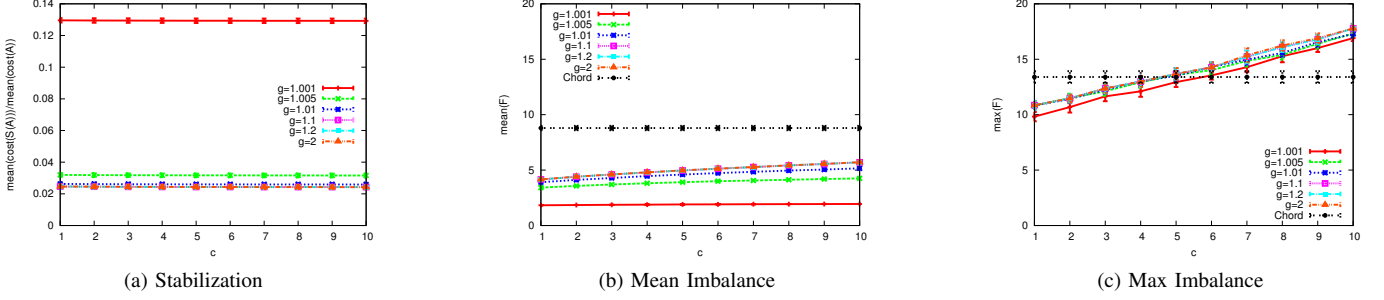


Fig. 2: Stabilization overhead (normalized by overhead of complete re-embedding) and corresponding imbalance of the content addressing of Algorithm 1 for various values of c and g

participate in the local re-embedding caused by the departure. Then, we have $\mathbb{E}(X_1) = \mathcal{O}(np)$. A node v has to participate if one of v 's descendants departs or one of v 's ancestors re-embeds. By Lemma V.1, the probability $P(E_1)$ that a descendant departs is $P(E_1) = \mathcal{O}(D/n)$. For the second event E_2 of being affected by an ancestor's re-embedding, we first consider the frequency of relayed embedding requests. Let Z denote the number of topology changes until a node u has to relay a request to its parent after the last re-embedding of u 's subtree initiated by any ancestor of u . In the following, we show that $\mathbb{E}(Z) = \Omega\left(\frac{n}{D^2}\right)$ regardless of u 's position in the tree. Thus, the probability that a node departure results in a re-embedding request from u to its parent is $P(E_3) = \mathcal{O}\left(\frac{D^2}{n}\right)$ for all nodes u . Now, a node v 's ancestor re-embeds if v or one of v 's ancestors or their siblings request a re-embed. v has at most D ancestors with an expected number of siblings of $\mathbb{E}(S)$, hence by a union bound $P(E_2) = P(E_3)DK = \frac{D^3(\mathbb{E}(S)-1)}{n}$. Thus,

$$\mathbb{E}(X_1) = \mathcal{O}(n(P(E_1) + P(E_2))) = \mathcal{O}(D^3\mathbb{E}(S)).$$

It remains to determine $\mathbb{E}(Z)$. We start by considering the number of descendants U of u that either depart or have to re-join due to a departure. Then, we derive a lower bound θ on the number of nodes that can depart before u relays a re-embedding request. It follows $\mathbb{E}(Z) = \theta/\mathbb{E}(U)$. The network size does not change considerably between a re-embedding from an ancestor and a request for re-embedding, as otherwise the root would initialize a re-embedding.

The probability that a departing node is one of u 's initial descendants is $\mathcal{O}\left(\frac{|V_u|}{n}\right)$. By Lemma V.1, a departure affects on average $\mathcal{O}(D)$ nodes, namely all descendants of the departing node. By conditioning on the fact that the departure takes place in subtree of potentially less than n nodes, $\mathcal{O}(D)$ remains a valid upper bound. Hence $\mathbb{E}(U) = \mathcal{O}\left(\frac{D|V_u|}{n}\right)$.

After a re-embedding initiated by $p(u)$ or another ancestor, we know that the content addressable storage is $(f(p(u)), \delta)$ -balanced with regard to V_u . Hence, by Eq. 5 and $\text{level}(p(u)) = \text{level}(u) - 1$, $\text{con}(V_u) = \sum_{v \in V_u} \mu(\mathbf{B}(v)) \leq$

$$|V_u| \left(\frac{g(1+c+\text{level}(u)-1)}{n_{\text{est}}g} + \delta \right) \text{ or}$$

$$|V_u| \geq \frac{\text{cont}(V_u)n_{\text{est}}g}{(1 - \frac{g}{f(u)})f(u) + n_{\text{est}}g\delta} \quad (10)$$

By Eq. 4, u has to request a re-embedding if the subtree size $|V'_u|$ has become so small that

$$|V'_u| \leq \frac{\text{cont}(V_u)n_{\text{est}}g}{f(u)} \quad (11)$$

As δ is negligible, $f(u) = \mathcal{O}(D)$, Eq. 10 and Eq. 11 result in $|V'_u| = \mathcal{O}\left(1 - \frac{1}{D}|V_u|\right)$. In other words, at least $\theta = \Omega\left(\frac{1}{D}|V_u|\right)$ descendants of u have to depart or re-join such that u has to re-embed. Hence,

$$\mathbb{E}(Z) \geq \frac{\theta}{\mathbb{E}(A)} = \Omega\left(\frac{|V_u|}{D} \frac{n}{D|V_u|}\right) = \Omega\left(\frac{n}{D^2}\right)$$

and thus indeed $\mathbb{E}(X_1) = \mathcal{O}(D^3\mathbb{E}(S))$ and $\mathbb{E}(\text{cost}_{\text{depart}}(\mathbf{S}(\mathbf{A}))) = \mathcal{O}(D^3\mathbb{E}(S))$. ■

We now obtain the general result for a tree of logarithmic depth, considering both the original algorithm $\mathbf{S}(\mathbf{A})$ and the version $\mathbf{S}(\mathbf{A})_{\text{virt}}$ relying on virtual binary trees. For the latter, the depth is bound by $\mathcal{O}(D \log n)$ rather than D but the expected number of siblings is $\mathbb{E}(S) \leq 1$.

Corollary V.4. *Let (G, id, C, ca) be a content-addressable storage with a tree-based greedy embedding id . The depth of the spanning tree is at most $\mathcal{O}(\log n)$. With S denoting the number of siblings of a node, the algorithm $\mathbf{S}(\mathbf{A})$ maintains a $(\mathcal{O}(\log n), \delta)$ -balanced content-addressable storage at communication complexity*

$$\mathbb{E}(\text{cost}(\mathbf{S}(\mathbf{A}))) = \mathcal{O}(\log^3 n \mathbb{E}(S)).$$

Using virtual binary trees, the algorithm $\mathbf{S}_{\text{virt}}(\mathbf{A})$ maintains a $(\mathcal{O}(\log^2 n), \delta)$ -balanced content-addressable storage at communication complexity

$$\mathbb{E}(\text{cost}(\mathbf{S}_{\text{virt}}(\mathbf{A}))) = \mathcal{O}(\log^6 n).$$

Thus, if the trees are reasonable regular, i.e., $\mathbb{E}(S)$ is bound by a constant or a (poly-)logarithmic factor, the original version $\mathbf{S}(\mathbf{A})$ exhibits both a better balance and lower computation complexity. However, if $\mathbb{E}(S)$ is high, i.e., there are few nodes with many children while the majority of nodes have few children, the computation complexity increases. Then,

the virtual binary tree version can offer a lower computation complexity at the price of a higher balance factor.

In the following, we simulate the actual overhead of the two algorithms and compare the results to these bounds.

VI. SIMULATIONS

We substantiate the previous asymptotic bounds by a simulation study considering the case of F2F overlays. Our goal is to provide concrete bounds on the stabilization complexity and the balance of the content addressing of Algorithm 1 for various values of the tree depth offset c and network size estimation accuracy g . Furthermore, we evaluate the impact of the simple join and virtual binary tree variant described in Section IV-C. In the following, we describe our simulation model, set-up, expectations and results.

Simulation Model and Set-up: Our simulation builds on GTNA [18], a framework for graph analysis. Aside from the parameters c and g , the performance of Algorithm 1 depends on the graph G and the churn pattern, i.e., the node join and departure sequence. In our simulation model, we characterize the latter by the session and intersession length distributions L_S and L_I . Furthermore, we use the spanning tree construction by Perlman [19], which assigns each node a random numerical identifier and then constructs a spanning tree of minimal depth such that the root corresponds to the node with the highest identifier.

During the set-up phase, each node chooses its random identifier for the spanning tree construction, which remains constant during the simulation. Initially, each node is online with probability $\frac{\mathbb{E}(L_S)}{\mathbb{E}(L_S + L_I)}$. If a node is online at start-up, we assume that it is at a random point of its current session. In other words, we choose the time of an online node's departure by selecting a session length l according to L_S and multiplying l with a uniformly chosen random number in $[0, 1)$. Analogously, we select the time until an offline node joins. Then, we execute the spanning tree construction on the subgraph induced by all initially online nodes. Subsequently, we execute Algorithm 2 on the same subgraph to obtain the initial embedding. If the graph is partitioned into multiple components, we execute the two algorithms for each component individually.

In each step i of the algorithm, we add or remove a node according to the previously selected sessions and intersession times. We choose the time of this node's next join or departure by selecting an interval l according to L_I or L_S , respectively, and add l to the currently elapsed time. Afterwards, we re-establish the spanning tree, potentially merging trees if previously disjointed components are connected or constructing new trees if new partitions are created. Last, we execute Algorithm 1, starting from either a newly joined node or the parent and children of a departed node.

During the simulation, we measure the number of all messages $cost_i(\mathbf{S}(\mathbf{A}))$ required for stabilization at step i . Let $mean(cost(\mathbf{S}(\mathbf{A})))$ denote the mean of $cost_i(\mathbf{S}(\mathbf{A}))$ over all i . Furthermore, we compare the cost of our algorithm to that of re-embedding at each topology change, i.e., we compute

$\frac{mean(cost(\mathbf{S}(\mathbf{A})))}{mean(\mathbf{A})}$ with $mean(\mathbf{A})$ being the mean number of messages to i) inform the root of the joined or departed node's tree of the change, and ii) executing Algorithm 2 on the complete tree. The quantity $\frac{mean(cost(\mathbf{S}(\mathbf{A})))}{mean(\mathbf{A})}$ thus indicates how much our changes reduce the stabilization complexity. In order to characterize the balance of the content addressing, we consider the fraction of addresses $\mu_i(u)$ assigned to each online node u in step i in relation to the number of nodes $n_i(u)$ in u 's component, i.e., we derive $F_{u,i} = \mu_i(u) \cdot n_i(u)$. For each step i , we derive the maximum $F_i = \max_{u:online} F_{u,i}$ and compare it to the upper bound on the maximal permitted imbalance defined in Eq. 5. We then consider the mean and maximal imbalance $mean(F)$ and $max(F)$ over all steps i .

Our sample set-up considers the case of F2F overlays, which are route-restricted and limit direct communication to devices of users with a mutual trust relationship. In this study, we use the friendship graph of a university online social network (SPI) of 9,222 students with an average of 10.58 connections to model trust relations [20]. SPI represents a friendship network and is thus in the absence of actual F2F overlay topologies a suitable model for such an overlay. In contrast to subgraphs of Facebook or other large-scale social networks, the locality of the network indicates that people sharing a link indeed share a real-world friendship or at least acquaintance. In order to judge the impact of the graph topology on the algorithms, we also generated one synthetic graph according to the model of Barabasi-Albert (BA) and another graph according to the model of Erdos-Renyi (ER-1), both with the same number of nodes and the same average degree of 10.58 as SPI. Furthermore, we generated another Erdos-Renyi graph with 9,222 nodes but a higher average degree of 922.2 (ER-2), to shed light on the impact of the density of the network. Our churn patterns follow the empirical session and intersession length measured in Freenet, an anonymous content sharing network with a F2F mode [21]. Based on these churn patterns, the number of concurrently online nodes usually varies between 3,700 and 4,000. As for the parameters of Algorithm 1, we varied c between 1 and 10 and choose $g \in \{1.001, 1.005, 1.01, 1.1, 1.2, 2\}$. All parameter combinations were considered for the original form of Algorithm 1 as well as for the simple join and the virtual binary tree variant. We averaged our result over 20 runs and present them with 95% confidence intervals. Each run consists of 100,000 consecutive node joins or departures. Note that we used the same 20 joins and departure sequences for each set of parameters in order to facilitate comparisons. For comparison, we also measure $mean(F_i)$ and $max(F_i)$ for a Chord overlay [22] of the same size using these join and departure sequences.

Expectations: Our expectations with regard to the parameters c and g on the stabilization overhead and the balance of the content addressing are governed by Eq. 5 and Propositions V.2 and V.3. Eq. 5 indicates that the upper bound on the maximal imbalance increases with both g and c . However, considering Line 15 of Algorithm 1, we see that g does not affect the actual decision of re-embedding. Rather, it only affects the certainty of nodes in the current network size estimation and thus has

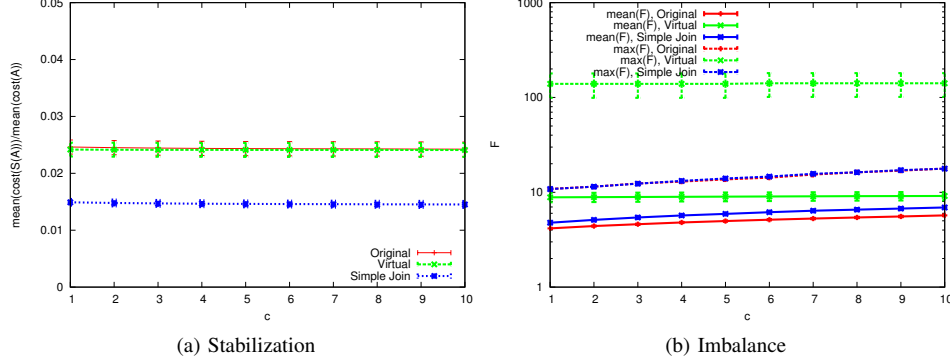


Fig. 3: Stabilization overhead and corresponding imbalance of the content addressing of Algorithm 1 in this original form, the virtual binary tree variant, and the simple join variant, $g = 2$

at most an indirect effect on the actual balance F_i . In contrast, c affects the decision in Line 15 and allows nodes to accept a larger imbalance. Hence, we expect an increase in $mean(F)$ and $max(F)$ with an increased c .

With regard to the stabilization complexity, the asymptotic bounds in Propositions V.2 and V.3 are independent of both g and c . Indeed, g only indicates the frequency of re-embeddings due to an inaccurate network size estimation. For larger g , such re-embeddings should be rare, so that we do not expect a considerable impact of g on either the actual imbalance or the stabilization overhead. However, for very low $g = 1.001$ or $g = 1.005$, re-embeddings only require a change of less than 1% in size, so that re-embeddings actually impact the overall stabilization overhead. Thus, we assume that the stabilization overhead is higher for these g whereas $mean(F)$ and $max(F)$ decrease due to the frequent re-embeddings, which re-establish a perfectly balanced address assignment.

The goal of using a simplified join mechanism and virtual binary trees is to reduce the stabilization overhead. While Proposition V.3 indicates a reduced stabilization overhead when using virtual binary trees, the increased depth of the binary trees entails an increased upper bound on the permitted imbalance as by Eq. 5. Thus, it is likely that the actual values $mean(F)$ and $max(F)$ are higher than for Algorithm 1. Similarly, our simplified join mechanism reduces the stabilization overhead by not requiring re-embeddings of the complete subtree rooted at the parent. However, such re-embeddings after joins present the opportunity to improve the balance if previous departures have already changed the content assignment within the subtree considerably but not sufficiently for a re-embedding. Thus, the price for the reduced stabilization complexity is likely to be an increased actual imbalance though the bound on the permitted imbalance remains unaffected.

We choose the four types of graphs (SPI, BA, ER-1, ER-2) in order to ascertain that our expectations with regard to the impact of the node degree hold. For instance, we expect that the low-degree random graph ER-1 results in trees with a low number of children per node and hence a high depth. By Eq. 5, the high depth should correlate with a high imbalance. Due to the low number of children, we expect a comparable

low stabilization overhead for ER-1. For analogous reasons, we expect the opposite results, namely a high stabilization overhead and a well-balanced content addressing, for ER-2. The results for BA and SPI should moderate between those of the two random graphs.

Results: Our results with regard to the impact of parameters c and g agree with the above expectations and underline the asymptotic bounds with concrete values. Fig. 2 displays the results for the original version of Algorithm 1 on the SPI graph. Notably, our algorithm reduces the stabilization overhead to 2-3% of a complete re-embedding for $g \geq 1.01$, as shown in Fig. 2a. In absolute numbers, the average number of messages sent per step is slightly above 80 for a network of 3,000 to 4,000 online nodes. As expected, very low values of g considerably increase the stabilization overhead because the network size estimation has to be adjusted frequently. Fig. 2b shows that there are nodes in the network that are responsible for 4 to 6 times as many addresses as the average node in their component for $g \geq 1.01$. For $g = 1.001$, $mean(F)$ can be as low as 1.82 at the price of a high stabilization overhead. In the worst case, displayed in Fig. 2c, the imbalance increase to up to a factor 20. Note that the depth of the spanning tree varies between 20 and 30, so that the observed maximum is usually considerably lower than the theoretical upper bound. While the stabilization overhead is indeed not significantly impacted by c and g , an increase of c entails an increase in imbalance, as expected. In comparison to Chord, a supposedly well-balanced P2P overlay, we achieve a lower value of $mean(F)$ for all considered parameters. With regard to $max(F)$, we achieve a higher degree of balance for $c < 5$. Hence, our content addressing achieves a similar or even better balance than existing solutions for content addressing.

Now, we consider the impact of protocol variants on stabilization and content addressing. Fig. 3 contrasts virtual binary trees and simple join with the original algorithm for $g = 2$ and $c = 1..10$. Indeed, both variants decrease the stabilization overhead further, as displayed in Fig. 3a. However, the insignificantly decreased stabilization overhead for the virtual tree variant comes at the price of a considerably higher imbalance. As the depth of the virtual tree is usually between 200 and 300, Fig. 3b shows that the actual observed imbalance

Sys	Original		Simple		Virtual	
SPI	0.025	4.16	0.015	4.78	0.024	8.84
BA	0.041	4.20	0.021	4.79	0.040	11.33
ER-1	0.023	7.72	0.020	8.58	0.022	27.91
ER-2	0.071	1.14	0.036	1.21	0.071	1.14

TABLE I: Stabilization overhead $\frac{mean(S(A))}{mean(A)}$ (left column) vs. mean imbalance factor $mean(F)$ (right column) of Alg. 1 ($c = 1$, $g = 2$) in different topologies: real-world social network SPI; Barabasi-Albert (BA); Erdos-Renyi, average degree 10.58 (ER-1); Erdos-Renyi, average degree 922.2 (ER-2)

can reach values close to 200. In contrast, a simple join only slightly increases $mean(F)$ but leaves $max(F)$ largely unaffected. Thus, the simulation study indicates that the actual improvement of the virtual binary tree variant with regard to stabilization overhead does not outweigh the drastic decrease in balance, at least for the considered graphs. However, using a simple join mechanism reduces the stabilization overhead by roughly a factor 2 without severe consequences on the balance of the content addressing.

Last, Table I displays $\frac{mean(S(A))}{mean(A)}$ and $mean(F)$ for the four considered topologies focusing on the case of $c = 1$, $g = 2$. The structure of the underlying graph drastically impacts the actual results. The stabilization overhead increases drastically for the densely connected network ER-2. Note that the virtual binary trees cannot counteract this increase, as the re-embedding is almost always executed by the parent of the newly joined or departed node. Thus, a simple join variant indeed nearly halves the overhead. Due to the low depth of the tree, ER-2 exhibits an extremely low imbalance $mean(F)$. In contrast, ER-1 exhibits a very low stabilization overhead but a considerably higher imbalance, as expected due to the low degree and high tree depth. BA and SPI, having a scale-free degree distribution with some high-degree nodes and mostly low-degree nodes, moderate between the extremes.

This evaluation complements our theoretical bounds with concrete numbers. Not only do these concrete numbers validate our theoretical bounds, they also indicate that our algorithm is efficient and can achieve a more balanced content addressing than commonly used content addressing schemes.

VII. CONCLUSION

The main contribution of this paper is the design and formal verification of an approach that efficiently generates tree-based greedy embeddings for balanced content addressing on fully dynamic networks. We realized our solution by designing a stabilization algorithm and an embedding algorithm where the former makes use of the latter to dynamically update content network addresses.

We proved that our approach guarantees fair distribution of content addresses and we showed that the expected cost of a single change in the network is logarithmic for node joins and polylogarithmic for node departures. Finally we confirmed these formal bounds in a simulation on realistic problem instances.

Future work may explore alternative implementations of our algorithms to potentially improve upon the complexity bounds

and the real world latency of both routing and stabilization.

ACKNOWLEDGEMENTS

This work in parts was supported by DFG through the CRC HAEC and the Cluster of Excellence cfaed.

REFERENCES

- [1] S. Roos, L. Wang, T. Strufe, and J. Kangasharju. Enhancing Compact Routing in CCN with Prefix Embedding and Topology-Aware Hashing. In *MobiArch*, 2014.
- [2] Y. Jiang et al. A Distributed Routing for Wireless Sensor Networks with Mobile Sink Based on the Greedy Embedding. *Ad Hoc Networks*, 2014.
- [3] I. Clarke et al. Private Communication Through a Network of Trusted Connections: The Dark Freenet. *Network*, 2010.
- [4] C. H. Papadimitriou and D. Ratajczak. On a Conjecture Related to Geometric Routing. *Theor. Comput. Sci.*, 344(1), 2005.
- [5] A. Cvetkovski and M. Crovella. Hyperbolic Embedding and Routing for Dynamic Graphs. In *INFOCOM*, 2009.
- [6] J. Herzen, C. Westphal, and P. Thiran. Scalable Routing Easy as Pie: a Practical Isometric Embedding Protocol. In *ICNP*, 2011.
- [7] R. Kleinberg. Geographic Routing using Hyperbolic Space. In *INFOCOM*, 2007.
- [8] A. Hofer, S. Roos, and T. Strufe. Greedy Embedding, Routing and Content Addressing for Darknets. In *NetSys*, 2013.
- [9] S. Roos and M. Beck. Anonymous Addresses for Efficient and Resilient Routing in F2F Overlays. In *INFOCOM*, 2016.
- [10] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *PODC*, 2002.
- [11] D. Eppstein and M. T. Goodrich. Succinct Greedy Geometric Routing Using Hyperbolic Geometry. *IEEE Trans. Computers*, 60(11), 2011.
- [12] P. Maymounkov. Greedy Embeddings, Trees, and Euclidean vs. Lobachevsky Geometry. <https://www.pdos.lcs.mit.edu/~petar/papers/maymounkov-greedy-prelim.pdf>, 2006.
- [13] C. Westphal and G. Pei. Scalable Routing via Greedy Embedding. In *INFOCOM*, 2009.
- [14] H. Zhang and S. Govindaiah. Greedy Routing via Embedding Graphs onto Semi-metric Spaces. In *FAW-AAIM*, 2011.
- [15] J. Newsome and D. Song. GEM: Graph EMbedding for routing and data-centric storage in sensor networks without geographic information. In *SenSys*, 2003.
- [16] G. Chen, T. Qiu, and F. Wu. Insight Into Redundancy Schemes in DHTs. *The Journal of Supercomputing*, 43(2), 2008.
- [17] BD-CAT: Long version. <https://dl.dropboxusercontent.com/u/31759962/tech.pdf>.
- [18] B. Schiller and T. Strufe. GTNA 2.0-A Framework for Rapid Prototyping and Evaluation of Routing Algorithms. In *Summersim*, 2013.
- [19] R. Perlman. An algorithm for distributed computation of a spanningtree in an extended LAN. In *SIGCOMM*, 1985.
- [20] T. Paul et al. The Students' Portal of Ilmenau: A Holistic OSN's User Behaviour Model. In *PICCIT*, 2015.
- [21] S. Roos, B. Schiller, S. Hacker, and T. Strufe. Measuring Freenet in the Wild: Censorship-Resilience under Observation. In *PETS*, 2014.
- [22] I. Stoica et al. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. *ACM SIGCOMM CCR*, 31(4), 2001.